UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
10/724,269	11/26/2003	Dana Henriksen	26530.91	1261

47699 7590 11/08/2007

HAYNES AND BOONE, LLP
901 Main Street
Suite 3100
Dallas, TX 75202

EXAMINER	
MEONSKE, TONIA L	
ART UNIT	**PAPER NUMBER**
2181	
MAIL DATE	**DELIVERY MODE**
11/08/2007	PAPER

Please find below and/or attached an Office communication concerning this application or proceeding.

The time period for reply, if any, is set in the attached communication.

Office Action Summary	Application No. 10/724,269	Applicant(s) HENRIKSEN, DANA	
	Examiner Tonia L. Meonske	Art Unit 2181	

-- The MAILING DATE of this communication appears on the cover sheet with the correspondence address --

Period for Reply

A SHORTENED STATUTORY PERIOD FOR REPLY IS SET TO EXPIRE 3 MONTH(S) OR THIRTY (30) DAYS, WHICHEVER IS LONGER, FROM THE MAILING DATE OF THIS COMMUNICATION.

- Extensions of time may be available under the provisions of 37 CFR 1.136(a). In no event, however, may a reply be timely filed after SIX (6) MONTHS from the mailing date of this communication.
- If NO period for reply is specified above, the maximum statutory period will apply and will expire SIX (6) MONTHS from the mailing date of this communication.
- Failure to reply within the set or extended period for reply will, by statute, cause the application to become ABANDONED (35 U.S.C. § 133). Any reply received by the Office later than three months after the mailing date of this communication, even if timely filed, may reduce any earned patent term adjustment. See 37 CFR 1.704(b).

Status

1)☒ Responsive to communication(s) filed on *8/29/2007*.

2a)☒ This action is **FINAL**. 2b)☐ This action is non-final.

3)☐ Since this application is in condition for allowance except for formal matters, prosecution as to the merits is closed in accordance with the practice under *Ex parte Quayle*, 1935 C.D. 11, 453 O.G. 213.

Disposition of Claims

4)☒ Claim(s) *1-28* is/are pending in the application.

4a) Of the above claim(s) _____ is/are withdrawn from consideration.

5)☐ Claim(s) _____ is/are allowed.

6)☒ Claim(s) *1-28* is/are rejected.

7)☐ Claim(s) _____ is/are objected to.

8)☐ Claim(s) _____ are subject to restriction and/or election requirement.

Application Papers

9)☐ The specification is objected to by the Examiner.

10)☐ The drawing(s) filed on _____ is/are: a)☐ accepted or b)☐ objected to by the Examiner.

Applicant may not request that any objection to the drawing(s) be held in abeyance. See 37 CFR 1.85(a).

Replacement drawing sheet(s) including the correction is required if the drawing(s) is objected to. See 37 CFR 1.121(d).

11)☐ The oath or declaration is objected to by the Examiner. Note the attached Office Action or form PTO-152.

Priority under 35 U.S.C. § 119

12)☐ Acknowledgment is made of a claim for foreign priority under 35 U.S.C. § 119(a)-(d) or (f).

a)☐ All b)☐ Some * c)☐ None of:

1.☐ Certified copies of the priority documents have been received.

2.☐ Certified copies of the priority documents have been received in Application No. _____.

3.☐ Copies of the certified copies of the priority documents have been received in this National Stage application from the International Bureau (PCT Rule 17.2(a)).

* See the attached detailed Office action for a list of the certified copies not received.

Attachment(s)

1) ☐ Notice of References Cited (PTO-892)
2) ☐ Notice of Draftsperson's Patent Drawing Review (PTO-948)
3) ☐ Information Disclosure Statement(s) (PTO/SB/08) Paper No(s)/Mail Date _____.
4) ☐ Interview Summary (PTO-413) Paper No(s)/Mail Date. _____.
5) ☐ Notice of Informal Patent Application
6) ☐ Other: _____.

Claim Rejections - 35 USC § 103

1. The following is a quotation of 35 U.S.C. 103(a) which forms the basis for all obviousness rejections set forth in this Office action:

> (a) A patent may not be obtained though the invention is not identically disclosed or described as set forth in section 102 of this title, if the differences between the subject matter sought to be patented and the prior art are such that the subject matter as a whole would have been obvious at the time the invention was made to a person having ordinary skill in the art to which said subject matter pertains. Patentability shall not be negatived by the manner in which the invention was made.

2. Claims 1-28 are rejected under 35 U.S.C. 103(a) as being unpatentable over Hagan et al (U.S. Patent # 5,966,547), herein referred to as Hagan, in view of Parlante (Linked List Basics), and Catino (U.S. Patent # 5,319,778).

3. As per **claim 1**, Hagan teaches a method for implementing a global queue (See column 4, lines 1-4: A shared queue is the same as a global queue).

4. Hagan does not explicitly teach the use of a linked list with head and tail pointers. Hagan does teach where any type of standard queue/linked list can be used with his invention (See column 3, lines 29-32).

5. Parlante does teach a linked list with head pointer (A head pointer is inherent with linked lists) and tail pointer (See section 3, part 4: Tail pointers are common with linked lists and thus is considered a basic of linked lists.). The queue would comprise:

a. the head pointer functioning as a next pointer of a last element of the plurality of elements when the queue is empty (See page 5: This normally happens in the boundary case and is well known in the art.); and

b. executing an add to end function for adding a new element to the end of the queue even when the queue is in a locked state in which the queue head

pointer is hull and a queue tail pointer does not point to the queue head pointer (See page 5: This is a normal occurrence with a linked list system, depending on the algorithm used, in which an element is in the process of being added to an empty queue. This in-between step will now be interpreted as being in locked state. It is also alluded by the teachings of the boundary case.), the executing an add to end function including setting a next pointer of the new element to null; as an atomic transaction, setting the queue tail pointer to point the new element while saving a location of the last element; and setting the next pointer of the last element to point to an address of the new element by using the last element's saved location (See section 3, part 4: This action is also inherent to queues and the actions are indicative of an enqueue to an empty queue).

6. It would have been obvious to a person have ordinary skill in the art at the time the invention was made to have modified Hagan with Parlante. Hagan already discloses the use of other queues (See column 3, lines 29-32). Parlante teaches the use of a linked list with head and tail pointers. Modifying the queue of Hagan wherein the queue has a head pointer, a tail, and a plurality of elements each having a next pointer would be a stylistic preference and is within the scope of the invention by Hagan.

7. Hagan and Parlante do not teach "adding a new element to the end of the queue even when the queue is in a locked state immediately prior to execution of the add to end function."

8. Catino does teach "adding a new element to the queue even when the queue is in a locked state immediately prior to execution of the add to end function." Such disclosure can be found in column 7, lines 25-47. The head pointer is in a locked state and the end pointer has a "potentially lockable" state, which allows for enqueue in a locked state while maintain integrity of data and the queue.

9. It would have been obvious to a person having ordinary skill in the art at the time the invention was made to have modified Hagan and Parlante to include the teachings of Catino. Parlante already teaches boundary cases (See page 5) in which a locked state exists. In the teachings of Parlante, enqueue has to wait until a locked state is vacated. Catino teaches where it is possible to enqueue during a locked state. According to the abstract of Hagan, "an efficient posting of entries to the queue" is a goal of the invention. Catino teaches such a goal and thus it would be obvious to one having ordinary skill in the art to modify Hagan in view of Parlante with the teachings of Catino.

10. As per **claim 2**, Hagan teaches further comprising selectively executing a locking function of the queue, the locking function including: if the previous value of the head pointer is null and the queue is not empty, repeating the locking function (See column 5, lines 55-57).

11. Hagan is silent on many other aspects of the queue but does teach where any type of standard queue/linked list can be used with his invention (See column 3, lines 29-32).

12. Parlante teaches if the queue is not empty and not locked, as an atomic transaction, setting the head pointer to null and retaining a previous value of the head pointer (See section 3, part 3).

13. It would have been obvious to a person have ordinary skill in the art at the time the invention was made to have modified Hagan to include the teachings of Parlante because although Hagan does not teach a linked list with a head and tail pointer, it is within the scope of the invention to include the teachings of Parlante.

14. As per **claim 3**, Hagan teaches a method for implementing a global queue (See column 4, lines 1-4: A shared queue is the same as a global queue).

15. Hagan is silent on a linked list but does teach where any type of standard queue/linked list can be used with his invention (See column 3, lines 29-32).

16. Parlante teaches wherein the queue is unlocked in a first situation in which the head pointer is not null (By the contrapositive of the definition in claim 1, this must be true).

17. Catino teaches wherein the queue is unlocked in a second situation in which the head pointer is null and the tail pointer points to the head pointer (column 7, lines 25-47: The queue is not locked since there is no elements in the queue and the tail pointer is locked only when there is one element in the queue).

18. It would have been obvious to a person have ordinary skill in the art at the time the invention was made to have modified Hagan to include the teachings of Parlante because although Hagan does not teach a linked list with a head and tail pointer, it is within the scope of the invention to include the teachings of Parlante.

19. It would have been obvious to a person having ordinary skill in the art at the time the invention was made to have modified Hagan and Parlante to include the teachings of Catino. Parlante already teaches boundary cases (See page 5) in which a locked state exists. In the teachings of Parlante, enqueue has to wait until a locked state is vacated. Catino teaches where it is possible to enqueue during a locked state. According to the abstract of Hagan, "an efficient posting of entries to the queue" is a goal of the invention. Catino teaches such a goal and thus it would be obvious to one having ordinary skill in the art to modify Hagan in view of Parlante with the teachings of Catino.

20. As per **claim 4**, Hagan teaches a method for implementing a global queue (See column 4, lines 1-4: A shared queue is the same as a global queue).

21. Hagan is silent on a linked list but does teach where any type of standard queue/linked list can be used with his invention (See column 3, lines 29-32).

22. Parlante teaches further comprising selectively executing an add to front function for adding the new element to a front position of the queue, the add to front function including: if the queue is empty adding the new element to an end position of the queue (This action is inherent to queues and the actions are indicative of an enqueue to an

empty queue); and if the queue is not empty: locking the queue; setting the next pointer of the new element to the previous value of the head pointer; and pointing the head pointer to the new element, thereby unlocking the queue (This action is inherent to queues).

23. It would have been obvious to a person have ordinary skill in the art at the time the invention was made to have modified Hagan to include the teachings of Parlante because although Hagan does not teach a linked list with a head and tail pointer, it is within the scope of the invention to include the teachings of Parlante.

24. As per **claim 5**, Hagan teaches a method for implementing a global queue (See column 4, lines 1-4: A shared queue is the same as a global queue).

25. Hagan is silent on a linked list but does teach where any type of standard queue/linked list can be used with his invention (See column 3, lines 29-32).

26. Parlante teaches further comprising executing a remove from front function to remove a front-most element from the queue, the remove from front function including: locking the queue (Inherently done to avoid conflicts in a multi-processor system); if the queue is not empty and an element occupying a front-most position of the queue has a next pointer that is not null, setting the head pointer to the address in the front-most element's next pointer (Inherently done with a dequeue so rest of queue is not lost); and if the queue is not empty and the front-most element's next pointer is null, as an atomic compare and exchange, if the tail pointer points to the front-most element, pointing the

tail pointer to the head pointer, thereby implicitly unlocking the queue (Inherently done with a dequeue with en empty queue).

27. It would have been obvious to a person have ordinary skill in the art at the time the invention was made to have modified Hagan to include the teachings of Parlante because although Hagan does not teach a linked list with a head and tail pointer, it is within the scope of the invention to include the teachings of Parlante.

28. As per **claim 6**, Hagan discloses further comprising, responsive to a failure of the atomic compare and exchange, waiting for the next pointer of the front-most element to become non-null, and pointing the head pointer to an element pointed to by the next pointer of the front-most element, thereby implicitly unlocking the queue (See column 5, lines 55-57).

29. As per **claim 7**, Hagan teaches a method for implementing a global queue (See column 4, lines 1-4: A shared queue is the same as a global queue).

30. Hagan is silent on a linked list but does teach where any type of standard queue/linked list can be used with his invention (See column 3, lines 29-32).

31. Parlante teaches further comprising executing a remove specific function to remove a target element from the queue, the remove specific function including: locking the queue (Inherently done to avoid conflicts in a multi-processor system); and if the queue is not empty: traversing the queue to locate the target element (Inherent as the only known elements are the first and last elements); if the target element's next pointer

is not null and the target is not addressed by the previous value of the head pointer, setting the next pointer of an element previous to the target to point to an element pointed to by the target's next pointer (Inherently done with a dequeue so rest of queue is not lost); and returning the head pointer to the previous value, thereby implicitly unlocking the queue (Inherently done with a dequeue).

32. It would have been obvious to a person have ordinary skill in the art at the time the invention was made to have modified Hagan to include the teachings of Parlante because although Hagan does not teach a linked list with a head and tail pointer, it is within the scope of the invention to include the teachings of Parlante.

33. As per **claim 8**, Hagan teaches a method for implementing a global queue (See column 4, lines 1-4: A shared queue is the same as a global queue).

34. Hagan is silent on a linked list but does teach where any type of standard queue/linked list can be used with his invention (See column 3, lines 29-32).

35. Parlante teaches further comprising: if the target element's next pointer is not null and the target is addressed by the previous value of the head pointer, setting the head pointer to point to the element pointed to by the target's next pointer, thereby implicitly unlocking the queue (Inherently done with a dequeue to return from its locked state); and if the target's next pointer is null and the target is not addressed by the previous value of the head pointer, setting the next pointer of the element previous to the target to null (Inherent done with a dequeue in which it element is to be removed).

36. It would have been obvious to a person have ordinary skill in the art at the time the invention was made to have modified Hagan to include the teachings of Parlante because although Hagan does not teach a linked list with a head and tail pointer, it is within the scope of the invention to include the teachings of Parlante.

37. As per **claim 9**, Hagan discloses further comprising: if the atomic compare and exchange was performed and failed: waiting until the target's next pointer is not null (See column 5, lines 55-57).

38. Hagan is silent on many other aspects of the queue but does teach where any type of standard queue/linked list can be used with his invention (See column 3, lines 29-32).

39. Parlante teaches if the target's next pointer is null, as an atomic compare and exchange, if the tail pointer points to the target setting the tail pointer to point to the element previous to the target (This is inherently done after a dequeue), or to point to the head pointer if the target is addressed by the previous value of the head pointer (This is inherent with an empty queue); if an element addressed by the target's next pointer is an only remaining element in the queue, sending the head pointer to point to the only remaining element, thereby implicitly unlocking the queue (This is inherent with actions after a dequeue at the top of the queue); and if the element addressed by the target's next pointer is not the only remaining element in the queue, setting the next pointer of the element previous to the target to the address in the target's next pointer and setting the head pointer to the previous value of the head pointer, thereby implicitly

unlocking the queue (This is inherent to ensure that the head and the tail point to the correct places); and if the atomic compare and exchange was performed and succeeded: if the queue is not empty, setting the head pointer to the previous value of the head pointer, thereby implicitly unlocking the queue (This is inherent with a normal dequeue).

40. It would have been obvious to a person have ordinary skill in the art at the time the invention was made to have modified Hagan to include the teachings of Parlante because although Hagan does not teach a linked list with a head and tail pointer, it is within the scope of the invention to include the teachings of Parlante.

41. As per **claim 10**, Hagan teaches a method for implementing a global queue (See column 4, lines 1-4: A shared queue is the same as a global queue).

42. Hagan is silent on a linked list but does teach where any type of standard queue/linked list can be used with his invention (See column 3, lines 29-32).

43. Parlante teaches further comprising executing an empty function for removing each of the plurality of elements from the queue, the empty function including: locking the queue; and if the queue is not empty: as an atomic transaction, pointing the tail pointer to the head pointer while retaining a previous value of the head pointer and the tail pointer, thereby implicitly unlocking the queue; and by using the previous values of the head pointer and tail pointer, traversing a plurality of the elements which have been dequeued, and waiting for the next pointer of each element not addressed by the previous value of the tail pointer to become non-null (This sort of action describes the

action of a deconstructor, which, if not inherent, is obvious to a person have ordinary skill in the art at the time the invention was made).

44. It would have been obvious to a person have ordinary skill in the art at the time the invention was made to have modified Hagan to include the teachings of Parlante because although Hagan does not teach a linked list with a head and tail pointer, it is within the scope of the invention to include the teachings of Parlante.

45. As per **claim 11**, Hagan teaches a method for implementing a global queue (See column 4, lines 1-4: A shared queue is the same as a global queue) in a multiprocessor environment (See column 2, lines 28-30), the method comprising: allowing a first processor to both add and remove elements from the queue (See abstract and column 5, lines 3-6 and 17-19: The abstract says one of the processors only posts entries in the queue and that processor is known as the posting processor. Thus the first processor would be the host processor, which can remove the entries. It later goes on to disclose both processors can post to the queue), allowing a second processor to only add new elements to the queue (See abstract and column 5, lines 3-6: The abstract says one of the processors only posts entries in the queue and that processor is known as the posting processor);

46. Hagan does not explicitly teach the use of a linked list with head and tail pointers. Hagan does teach where any type of standard queue/linked list can be used with his invention (See column 3, lines 29-32).

47. Parlante does teach a queue which has a head pointer to point to a first element of the queue or to null if the queue is empty (A head pointer is inherent with linked lists), a tail pointer to point to a last element of the queue or to the head pointer if the queue is empty (See section 3, part 4: Tail pointers are common with linked lists and thus is considered a basic of linked lists), and a plurality of elements each containing a next pointer for pointing to a next element in the queue or to null when the element occupies a last position in the queue (Inherent to queues known as linked lists), the head pointer functioning as a next pointer of the last element of the queue when the queue is empty (See page 5: This normally happens in the boundary case and is well known in the art), the method comprising:

a. executing an add to end function for adding the new element to the end of the queue, even when the queue is in a locked state in which the queue head pointer is null and a queue tail pointer does not point to the queue head pointer (See page 5: This can a normal occurrence with a linked list system, depending on the algorithm used, in which an element is in the process of being added to an empty queue. This in-between step will now be interpreted as being in locked state. It is also eluded by the teachings of the boundary case),

b. wherein the executing an add to end function includes setting the next pointer of the new element to null; as an atomic transaction, setting the tail pointer to point the new element, while saving a location of the last element; and setting the next pointer of the last element to point to the address of the new element by using the last element's saved location (See section 3, part 4: This

action is also inherent to queues and the actions are indicative of an enqueue to an empty queue).

48. It would have been obvious to a person have ordinary skill in the art at the time the invention was made to have modified Hagan with Parlante. Hagan already discloses the use of other queues (See column 3, lines 29-32). Parlante teaches the use of a linked list with head and tail pointers. Modifying the queue of Hagan wherein the queue has a head pointer, a tail, and a plurality of elements each having a next pointer would be a stylistic preference and is within the scope of the invention by Hagan.

49. Hagan and Parlante do not teach "adding a new element to the end of the queue even when the queue is in a locked state immediately prior to execution of the add to end function."

50. Catino does teach "adding a new element to the end of the queue even when the queue is in a locked state immediately prior to execution of the add to end function." Such disclosure can be found in column 7, lines 25-47. The head pointer is in a locked state and the end pointer has a "potentially lockable" state, which allows for enqueue in a locked state while maintain integrity of data and the queue.

51. It would have been obvious to a person having ordinary skill in the art at the time the invention was made to have modified Hagan and Parlante to include the teachings of Catino. Parlante already teaches boundary cases (See page 5) in which a locked state exists. In the teachings of Parlante, enqueue has to wait until a locked state is vacated. Catino teaches where it is possible to enqueue during a locked state.

According to the abstract of Hagan, "an efficient posting of entries to the queue" is a goal of the invention. Catino teaches such a goal and thus it would be obvious to one having ordinary skill in the art to modify Hagan in view of Parlante with the teachings of Catino.

52. As per **claim 12**, Hagan teaches a method for implementing a global queue (See column 4, lines 1-4: A shared queue is the same as a global queue).

53. Hagan is silent on a linked list but does teach where any type of standard queue/linked list can be used with his invention (See column 3, lines 29-32).

54. Parlante teaches further comprising executing an empty function for removing each element from the queue, the empty function including waiting until the head pointer is not null or until the queue is empty, and if the queue is not empty: saving a value of the head pointer; setting the head pointer to null; as an atomic transaction, pointing the tail pointer to the head pointer while saving a value of the tail pointer; and using the saved values of the head pointer and tail pointer, traversing the dequeued elements and waiting for the next pointer of each element not addressed by the saved value of the tail pointer to become non-null (This sort of action describes the action of a deconstructor, which, if not inherent, is obvious to a person have ordinary skill in the art at the time the invention was made).

55. It would have been obvious to a person have ordinary skill in the art at the time the invention was made to have modified Hagan to include the teachings of Parlante

because although Hagan does not teach a linked list with a head and tail pointer, it is within the scope of the invention to include the teachings of Parlante.

56. As per **claim 13**, Hagan teaches a method for implementing a global queue (See column 4, lines 1-4: A shared queue is the same as a global queue).

57. Hagan is silent on a linked list but does teach where any type of standard queue/linked list can be used with his invention (See column 3, lines 29-32).

58. Parlante teaches further comprising executing a remove from front function for removing a front-most element from the queue, the remove from front function including waiting until the head pointer is not null or until the queue is empty and if the queue is not empty; if the front-most element's next pointer is not null, setting the head pointer to an address of the front-most element's next pointer (Inherently done with a dequeue so rest of queue is not lost); if the front-most element's next pointer is null, as an atomic compare and exchange, if the tail pointer points to the front-most element, pointing the tail pointer to the head pointer (Inherently done with a dequeue with an empty queue).

59. It would have been obvious to a person have ordinary skill in the art at the time the invention was made to have modified Hagan to include the teachings of Parlante because although Hagan does not teach a linked list with a head and tail pointer, it is within the scope of the invention to include the teachings of Parlante.

60. As per **claim 14**, Hagan discloses further comprising responsive to a failure of the atomic compare and exchange, waiting for the next pointer of the front-most

element to become non-null, and pointing the head pointer to the element pointed to by the next pointer of the front-most element (See column 5, lines 55-57).

61. As per **claim 15**, Hagan discloses a system for implementing a global queue (See column 4, lines 1-4: A shared queue is the same as a global queue), the system comprising: a first processor (See column 2, lines 28-30: There are more than one processor); a plurality of instructions for execution on at least the first processor (See column 2, lines 50-54: Instructions are also inherent to processors), the instructions including instructions for: defining a locked state for the queue (See column 4, lines 47-54: Interrupts are used to lock the queue).

62. Hagan does not explicitly teach the use of a linked list with head and tail pointers. Hagan does teach where any type of standard queue/linked list can be used with his invention (See column 3, lines 29-32).

63. Parlante does teach wherein the queue has a head pointer to point to a first element of the queue or to null if the queue is empty (A head pointer is inherent with linked lists) and to function as a next pointer of a last element when the queue is emptily (See page 5: This normally happens in the boundary case and is well known in the art) [sic], a tail pointer to point to the last element of the queue or to the head pointer if the queue is empty (See section 3, part 4: Tail pointers are common with linked lists and thus is considered a basic of linked lists), and a plurality of elements each having a next pointer for pointing to a next element in the queue or to null when the element occupies a last position in the queue (Linked lists inherently have the next link), the system

comprising: executing a locked state for the queue (See page 5: This normally happens in the boundary case and is well known in the art); and executing an add at end function for adding a new element to the queue even when the queue is in a locked state in which the queue head pointer is null and a queue tail pointer does not point to the queue head pointer (See page 5: This can a normal occurrence with a linked list system, depending on the algorithm used, in which an element is in the process of being added to an empty queue. This in-between step will now be interpreted as being in locked state. It is also eluded by the teachings of the boundary case), the add at end function including setting the next pointer of the new element to null; as an atomic transaction, setting the tail pointer to point the new element, while saving a location of the last element; and setting the next pointer of the last element to point to the address of the new element by using the last element's saved location (See section 3, part 4: This action is also inherent to queues and the actions are indicative of an enqueue to an empty queue).

64. It would have been obvious to a person have ordinary skill in the art at the time the invention was made to have modified Hagan with Parlante. Hagan already discloses the use of other queues (See column 3, lines 29-32). Parlante teaches the use of a linked list with head and tail pointers. Modifying the queue of Hagan wherein the queue has a head pointer, a tail, and a plurality of elements each having a next pointer would be a stylistic preference and is within the scope of the invention by Hagan.

65. Hagan and Parlante do not teach “adding a new element to the end of the queue even when the queue is in a locked state immediately prior to execution of the add to end function.”

66. Catino does teach “adding a new element to the end of the queue even when the queue is in a locked state immediately prior to execution of the add to end function.” Such disclosure can be found in column 7, lines 25-47. The head pointer is in a locked state and the end pointer has a “potentially lockable” state, which allows for enqueue in a locked state while maintain integrity of data and the queue.

67. It would have been obvious to a person having ordinary skill in the art at the time the invention was made to have modified Hagan and Parlante to include the teachings of Catino. Parlante already teaches boundary cases (See page 5) in which a locked state exists. In the teachings of Parlante, enqueue has to wait until a locked state is vacated. Catino teaches where it is possible to enqueue during a locked state. According to the abstract of Hagan, “an efficient posting of entries to the queue” is a goal of the invention. Catino teaches such a goal and thus it would be obvious to one having ordinary skill in the art to modify Hagan in view of Parlante with the teachings of Catino.

68. As per **claim 16**, Hagan discloses further comprising: a second processor (See column 2, lines 28-30: There are more than one processor); wherein in the locked state only the first processor is allowed to remove elements from the queue (See column 4, lines 47-54: Interrupts are used to lock the queue).

69. As per **claim 17**, Hagan teaches a method for implementing a global queue (See column 4, lines 1-4: A shared queue is the same as a global queue).

70. Hagan is silent on a linked list but does teach where any type of standard queue/linked list can be used with his invention (See column 3, lines 29-32). Parlante teaches further comprising instructions for executing an empty function for removing each element from the queue, the empty function comprising: waiting until the head pointer is not null, or until the queue is empty; and if the queue is not empty: saving a value of the head pointer; setting the head pointer to null; as an atomic transaction, pointing the tail pointer to the head pointer while saving a value of the tail pointer; and using the saved values of the head pointer and tail pointer, traversing the dequeued elements and waiting for the next pointer of each element not addressed by the saved value of the tail pointer to become non null (This sort of action describes the action of a deconstructor, which, if not inherent, is obvious to a person have ordinary skill in the art at the time the invention was made).

71. It would have been obvious to a person have ordinary skill in the art at the time the invention was made to have modified Hagan to include the teachings of Parlante because although Hagan does not teach a linked list with a head and tail pointer, it is within the scope of the invention to include the teachings of Parlante.

72. As per **claim 18**, Hagan teaches a method for implementing a global queue (See column 4, lines 1-4: A shared queue is the same as a global queue).

73. Hagan is silent on a linked list but does teach where any type of standard queue/linked list can be used with his invention (See column 3, lines 29-32).

74. Parlante teaches further comprising instructions for executing a remove from front function for removing a front-most element from the queue, the remove from front function comprising waiting until the head pointer is not null or until the queue is empty, and if the queue is not empty; and if the front-most element's next pointer is not null, setting the head pointer to the address in the front-most element's next pointer (Inherently done with a dequeue so rest of queue is not lost); if the front-most element's next pointer is null, as an atomic compare and exchange, if the tail pointer points to the front-most element, pointing the tail pointer to the head pointer (Inherently done with a dequeue with an empty queue).

75. It would have been obvious to a person have ordinary skill in the art at the time the invention was made to have modified Hagan to include the teachings of Parlante because although Hagan does not teach a linked list with a head and tail pointer, it is within the scope of the invention to include the teachings of Parlante.

76. As per **claim 19**, Hagan discloses further comprising instructions for, responsive to a failure of the atomic compare and exchange, waiting for the next pointer of the front-most element to become non-null and pointing the head pointer to the element pointed to by the next pointer of the front-most element (See column 5, lines 55-57).

77. As per **claim 20**, Hagan teaches an implementation of a global queue (See column 4, lines 1-4: A shared queue is the same as a global queue).

78. Hagan is silent on a linked list but does teach where any type of standard queue/linked list can be used with his invention (See column 3, lines 29-32).

79. Parlante teaches wherein the queue is unlocked in a first situation in which the head pointer is not null (By the contrapositive of the definition in claim 1, this must be true).

80. Catino teaches wherein the queue is unlocked in a second situation in which the head pointer is null and the tail pointer points to the head pointer (column 7, lines 25-47: The queue is not locked since there is no elements in the queue and the tail pointer is locked only when there is one element in the queue).

81. It would have been obvious to a person have ordinary skill in the art at the time the invention was made to have modified Hagan to include the teachings of Parlante because although Hagan does not teach a linked list with a head and tail pointer, it is within the scope of the invention to include the teachings of Parlante.

82. It would have been obvious to a person having ordinary skill in the art at the time the invention was made to have modified Hagan and Parlante to include the teachings of Catino. Parlante already teaches boundary cases (See page 5) in which a locked state exists. In the teachings of Parlante, enqueue has to wait until a locked state is vacated. Catino teaches where it is possible to enqueue during a locked state. According to the abstract of Hagan, "an efficient posting of entries to the queue" is a goal of the invention. Catino teaches such a goal and thus it would be obvious to one

having ordinary skill in the art to modify Hagan in view of Parlante with the teachings of Catino.

83. As per **claim 21**, Hagan teaches wherein the instructions for executing a locked state for the queue comprise instructions for locking the queue when the head pointer is null and the tail pointer does not point to the head pointer (This can a normal occurrence with a linked list system, depending on the algorithm used, in which an element is in the process of being added to an empty queue. This in-between step will now be interpreted as being in locked state); and instructions for executing a locking function for the queue (See column 4, lines 47-54: Interrupts are used to lock the queue), the instructions for executing a locking function comprising: if the previous value of the head pointer is null and the queue is not empty repeating the locking function (See column 5, lines 55-57).

84. Hagan is silent on many other aspects of the queue but does teach where any type of standard queue/linked list can be used with his invention (See column 3, lines 29-32).

85. Parlante teaches if the queue is not empty and not locked, as an atomic transaction, setting the head pointer to null and retaining a previous value of the head pointer (See section 3, part 3).

86. It would have been obvious to a person have ordinary skill in the art at the time the invention was made to have modified Hagan to include the teachings of Parlante

because although Hagan does not teach a linked list with a head and tail pointer, it is within the scope of the invention to include the teachings of Parlante.

87. As per **claim 22**, Hagan teaches a method for implementing a global queue (See column 4, lines 1-4: A shared queue is the same as a global queue).
Hagan is silent on a linked list but does teach where any type of standard queue/linked list can be used with his invention (See column 3, lines 29-32).

88. Parlante teaches further comprising instructions for executing an add to front function, wherein the new element is added to a front position of the queue, the instructions for executing an add to front function comprising instructions for:

a. if the queue is empty, adding the new element to a last position of the queue (This action is inherent to queues and the actions are indicative of an enqueue to an empty queue); and if the queue is not empty: locking the queue; saving a previous value of the head pointer; setting the next pointer of the new element to the previous value of the head pointer; and pointing the head pointer to the new element, thereby unlocking the queue (This action is inherent to queues).

89. It would have been obvious to a person have ordinary skill in the art at the time the invention was made to have modified Hagan to include the teachings of Parlante because although Hagan does not teach a linked list with a head and tail pointer, it is within the scope of the invention to include the teachings of Parlante.

90. As per **claim 23**, Hagan teaches a method for implementing a global queue (See column 4, lines 1-4: A shared queue is the same as a global queue).

91. Hagan is silent on a linked list but does teach where any type of standard queue/linked list can be used with his invention (See column 3, lines 29-32).

92. Parlante teaches further comprising instructions for executing an empty function for removing each element from the queue, the instructions for executing an empty function comprising instructions for: locking the queue; and if the queue is not empty: as an atomic transaction, pointing the tail pointer to the head pointer while saving a value of the head and tail pointers, thereby implicitly unlocking the queue; and by using the saved values of the head pointer and tail pointer, traversing the dequeued elements and waiting for the next pointer of each element not addressed by the saved value of the tail pointer to become non null (This sort of action describes the action of a deconstructor, which, if not inherent, is obvious to a person have ordinary skill in the art at the time the invention was made).

93. It would have been obvious to a person have ordinary skill in the art at the time the invention was made to have modified Hagan to include the teachings of Parlante because although Hagan does not teach a linked list with a head and tail pointer, it is within the scope of the invention to include the teachings of Parlante.

94. As per **claim 24**, Hagan teaches a method for implementing a global queue (See column 4, lines 1-4: A shared queue is the same as a global queue).

95. Hagan is silent on a linked list but does teach where any type of standard queue/linked list can be used with his invention (See column 3, lines 29-32).

96. Parlante teaches further comprising instructions for executing a remove from front function for removing a front-most element from the queue, the instructions for executing a remove from front function comprising instructions for: locking the queue (Inherently done to avoid conflicts in a multi-processor system); if the queue is not empty and the front-most element's next pointer is not null, setting the head pointer to an address in the front-most element's next pointer (Inherently done with a dequeue so rest of queue is not lost); and if the queue is not empty and the front-most element's next pointer is null, as an atomic compare and exchange, if the tail pointer points to the front-most element, pointing the tail pointer to the head pointer, thereby implicitly unlocking the queue (Inherently done with a dequeue with en empty queue).

97. It would have been obvious to a person have ordinary skill in the art at the time the invention was made to have modified Hagan to include the teachings of Parlante because although Hagan does not teach a linked list with a head and tail pointer, it is within the scope of the invention to include the teachings of Parlante.

98. As per **claim 25**, Hagan discloses further comprising instructions for, responsive to performance and failure of the atomic compare and exchange, waiting for the next pointer of the front-most element to become non-null and pointing the head pointer to the element pointed to by the next pointer of the front-most element, thereby implicitly unlocking the queue (See column 5, lines 55-57).

99. As per **claim 26**, Hagan teaches a method for implementing a global queue (See column 4, lines 1-4: A shared queue is the same as a global queue).

100. Hagan is silent on a linked list but does teach where any type of standard queue/linked list can be used with his invention (See column 3, lines 29-32).

101. Parlante teaches further comprising instructions for executing a remove specific function for removing a target element from the queue, the instructions for executing a remove specific function comprising instructions for: locking the queue (Inherently done to avoid conflicts in a multi-processor system); determining if the queue is not empty; and if the queue is not empty: traversing the queue to locate the target element (Inherent as the only known elements are the first and last elements); and if the target element's next pointer is not null and the target element is not addressed by the previous value of the head pointer, setting the next pointer of an element previous to the target element to point to an element pointed to by the target element's next pointer (Inherently done with a dequeue so rest of queue is not lost), and return the head pointer to the previous value, thereby implicitly unlocking the queue (Inherently done with a dequeue).

102. It would have been obvious to a person have ordinary skill in the art at the time the invention was made to have modified Hagan to include the teachings of Parlante because although Hagan does not teach a linked list with a head and tail pointer, it is within the scope of the invention to include the teachings of Parlante.

103. As per **claim 27**, Hagan teaches a method for implementing a global queue (See column 4, lines 1-4: A shared queue is the same as a global queue).

104. Hagan is silent on a linked list but does teach where any type of standard queue/linked list can be used with his invention (See column 3, lines 29-32).

105. Parlante teaches further comprising instructions for: if the target's next pointer is not null and the target is net addressed by the previous value of the head pointer, setting the head pointer to point to the element pointed to by the target's next pointer, thereby implicitly unlocking the queue (Inherently done with a dequeue to return from its locked state); if the target's next pointer is null and the target is not addressed by the previous value of the head pointer, setting the next pointer of the element prior to the target to null (Inherent done with a dequeue in which it element is to be removed); and if the target's next pointer is null, as an atomic compare and exchange, if the tail pointer points to the target set the tail pointer to point to the element previous to the target (This is inherently done after a dequeue), or to point to the head pointer if the target is addressed by the previous value of the head pointer (This is inherent with an empty queue).

106. It would have been obvious to a person have ordinary skill in the art at the time the invention was made to have modified Hagan to include the teachings of Parlante because although Hagan does not teach a linked list with a head and tail pointer, it is within the scope of the invention to include the teachings of Parlante.

107. As per **claim 28**, Hagan discloses further comprising instructions for, responsive to performance and failure of the atomic compare and exchange:

a. waiting until the target's next pointer is not null (See column 5, lines 55-57). Hagan is silent on many other aspects of the queue but does teach where any type of standard queue/linked list can be used with his invention (See column 3, lines 29-32).

108. Parlante teaches if an element addressed by the target's next pointer is an only remaining element in the queue, setting the head pointer to point to the only remaining element, thereby implicitly unlocking the queue (Inherently done after a dequeue); if the element addressed by the target's next pointer is not the only remaining element in the queue, setting the next pointer of the element previous to the target to the address in the next pointer of the target and setting the head pointer to the previous value of the head pointer, thereby implicitly unlocking the queue (This is inherently done with a dequeue); and if the atomic compare and exchanged was performed and succeeded: if the queue is not empty setting the head pointer to the previous value of the head pointer, thereby implicitly unlocking the queue (This is inherently done with a normal dequeue).

109. It would have been obvious to a person have ordinary skill in the art at the time the invention was made to have modified Hagan to include the teachings of Parlante because although Hagan does not teach a linked list with a head and tail pointer, it is within the scope of the invention to include the teachings of Parlante.

Response to Arguments

110. Applicant's arguments filed August 29, 2007 have been fully considered but they are not persuasive.

111. On page 11, Applicant argues in essence:

> *"Catino's teaching of the head pointer in a locked state and the tail pointer in a "potentially lockable state" does not read on "the queue is in a locked state," which phrase is explicitly defined in the claims as comprising the state in which "the queue head pointer is null and the queue tail pointer does not point to the queue head pointer.""*

However, in response to applicant's arguments against the references individually, one cannot show nonobviousness by attacking references individually where the rejections are based on combinations of references. See *In re Keller*, 642 F.2d 413, 208 USPQ 871 (CCPA 1981); *In re Merck & Co.*, 800 F.2d 1091, 231 USPQ 375 (Fed. Cir. 1986). In this case, Parlante has been cited for having taught a queue in a locked state as defined by the claims (page 5). Therefore, the fact that Catino may not have taught a locked state where "the queue head pointer is null and the queue tail pointer does not point to the queue head pointer" is irrelevant. Therefore this argument is moot.

112. On page 12, Applicant argues in essence:

> *"The portion of Catino cited by the Examiner is presented in the context of adding an element to the beginning (or head) of the queue (see Catino, col. 6, 1.40-col. 8, 1. 46), whereas claim 1; as amended, expressly recites addition of an element to the end of the queue."*

However, Examiner respectfully notes the all queues have two ends, a head end and a tail end. So when an element in Catino is added to the beginning of the

queue, then an element is also being added to the end of the queue (see Catino, col. 6, 1.40-col. 8, 1. 46). Therefore this argument is moot.

Conclusion

113. **THIS ACTION IS MADE FINAL.** Applicant is reminded of the extension of time policy as set forth in 37 CFR 1.136(a).

114. A shortened statutory period for reply to this final action is set to expire THREE MONTHS from the mailing date of this action. In the event a first reply is filed within TWO MONTHS of the mailing date of this final action and the advisory action is not mailed until after the end of the THREE-MONTH shortened statutory period, then the shortened statutory period will expire on the date the advisory action is mailed, and any extension fee pursuant to 37 CFR 1.136(a) will be calculated from the mailing date of the advisory action. In no event, however, will the statutory period for reply expire later than SIX MONTHS from the mailing date of this final action.

115. Any inquiry concerning this communication or earlier communications from the examiner should be directed to Tonia L. Meonske whose telephone number is (571) 272-4170. The examiner can normally be reached on Monday-Friday with first Friday's off.

116. If attempts to reach the examiner by telephone are unsuccessful, the examiner's supervisor, Alford Kindred can be reached on (571) 272-4037. The fax phone number for the organization where this application or proceeding is assigned is 571-273-8300.

117. Information regarding the status of an application may be obtained from the Patent Application Information Retrieval (PAIR) system. Status information for published applications may be obtained from either Private PAIR or Public PAIR. Status information for unpublished applications is available through Private PAIR only. For more information about the PAIR system, see http://pair-direct.uspto.gov. Should you have questions on access to the Private PAIR system, contact the Electronic Business Center (EBC) at 866-217-9197 (toll-free). If you would like assistance from a USPTO Customer Service Representative or access to the automated information system, call 800-786-9199 (IN USA OR CANADA) or 571-272-1000.

TLM

Tonia L. Meonske

/Tonia L. Meonske/
Tonia L. Meonske
November 2, 2007